Design Review 2
ECE 3663
Spring 2013

Billl Devine
Frank Hiemstra
Ryan Hughes
Yatzek Krzepicki

Our approach to implementing the requirements of our design specifications will be detailed in the following design review, the second so far in this process. This review covers the second four functions (ADD, 2COMP, SHIFT, Register) as well as the top-level connections of the DSP block.

We outline our approach to the requirements further specified in design review 2, as well as the results of simulations we offer as evidence of functionality.

Our group has decided to create a multiplier block to serve as the arbitrary function in our ALU. We hope to utilize the adder block we created in designing this multiplier.

Progress & Remaining Tasks

Our team still needs to design and test the remaining ALU.

Upon completing this Design Review, we have seen improvements in both our team's efficiency in building and simulating using Cadence. All the individual parts we have made have been simulated and work; however, going forward we need to pay close attention to the area, power, and delays of the parts to optimize these parameters for IBL.

As far a testing goes, we first need to determine how all the individual parts will work in conjugation to the input and output registers. Along with that, we need to ensure that the ALU works under all varying input conditions. The results from our testing will allow us to understand what we need to change in order to maximize efficiency.

In order to decide the W/L for the transistors, we will test the entire ALU structure and varying the components in order to achieve the desired maximum efficiency in the parameter(s) we choose.

We understand that integrating all the parts together and the final testing is a crucial and difficult part of an embedded DSP. We intend to treat this difficult task "with respect" by starting early and using the Teaching Assistance resource if needed to carry out the task to complete the project.

By START of Week 4/22/13:

- Build and simulate Multiplier (Frank Hiemstra, Ryan Hughes)
- Fix power supply from 2.5 volts to 1.1 volts where necessary (team)
- Make definite choices about W/L for the transistors where necessary (team)
- Modify mux (Yatzek Krzepicki, Bill Devine)
- Manage parts to optimize power, delay, and area

By END of Week 4/22/13:

- Compile the entirety of the ALU (Frank Hiemstra, Yatzek Krzepicki)
- Finish Testing ALU (team)

By START of Week 4/29/13:

- Update online wiki
- Have a PowerPoint of the finished product
- Final Report completed and turned into IBL

Appendix A: Schematics of ALU Sub-Functions and Components

A.2.1: ADD

Schematic 2.1.a: Bit-Level Adder Implementation
Schematic 2.1.b: Adder Byte-Level Hierarchy

A.2.2: 2COMP

Schematic 2.2.a: 8-Bit hierarchy for 2COMP function

A.2.3: SHIFT

Schematic 2.3.a: Bit-Level Shift Implementation
Schematic 2.3.b: Shift Byte-Level Hierarchy

A.2.4: Register

Schematic 2.4.a: Bit-Level Register Implementation
Schematic 2.4.b: Register Byte-Level Hierarchy

A.2.5: Top-Level Connections

Schematic 2.5: Top-Level Block Connection Diagram

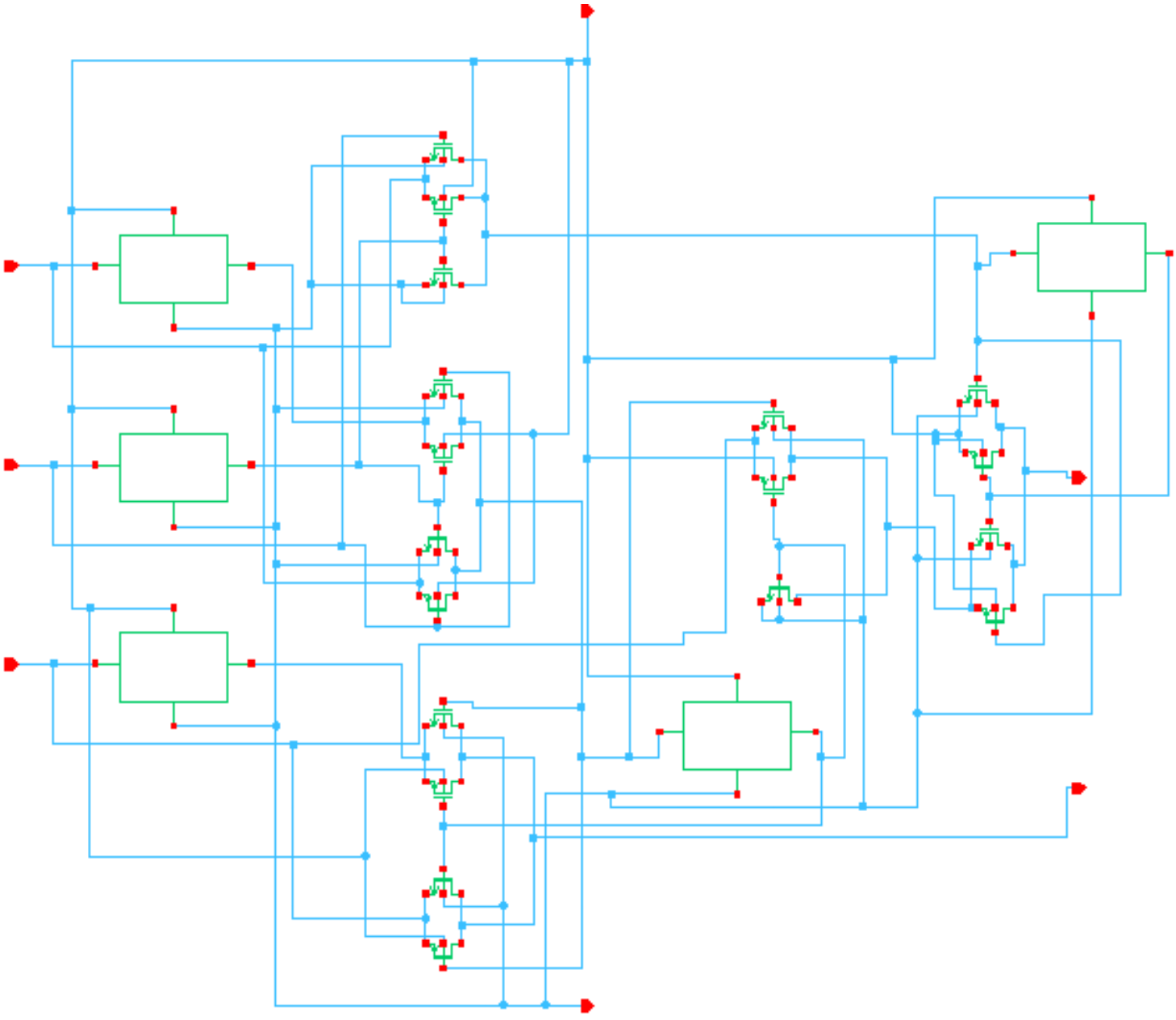
Appendix A.2.1: implementation schematics – Add

For the adder, we decided to go with a Pass Transistor Logic Carry Lookahead Adder. This uses a block, called the Carry Lookahead block, to predict carry outputs ahead of time, rather than alongside the sums. The logic used to do this is included below. We chose this topology because it typically results in lower loads at inputs and fewer transistors. The one issue that did arise was the failure of NMOS transistors to transmit a full logic 'high'. This was solved by using both a PMOS and an NMOS on paths which might transmit high signals. This solves this problem. In future optimization we may eliminate NMOS devices on paths which solely transmit high signals, just as we can eliminate PMOS devices on paths which only transmit low signals. The transistor level schematic for a single full adder, including the carry lookahead logic, is included below (2.1.a), as is the block level layout of the full 8 bit adder (2.1.b).

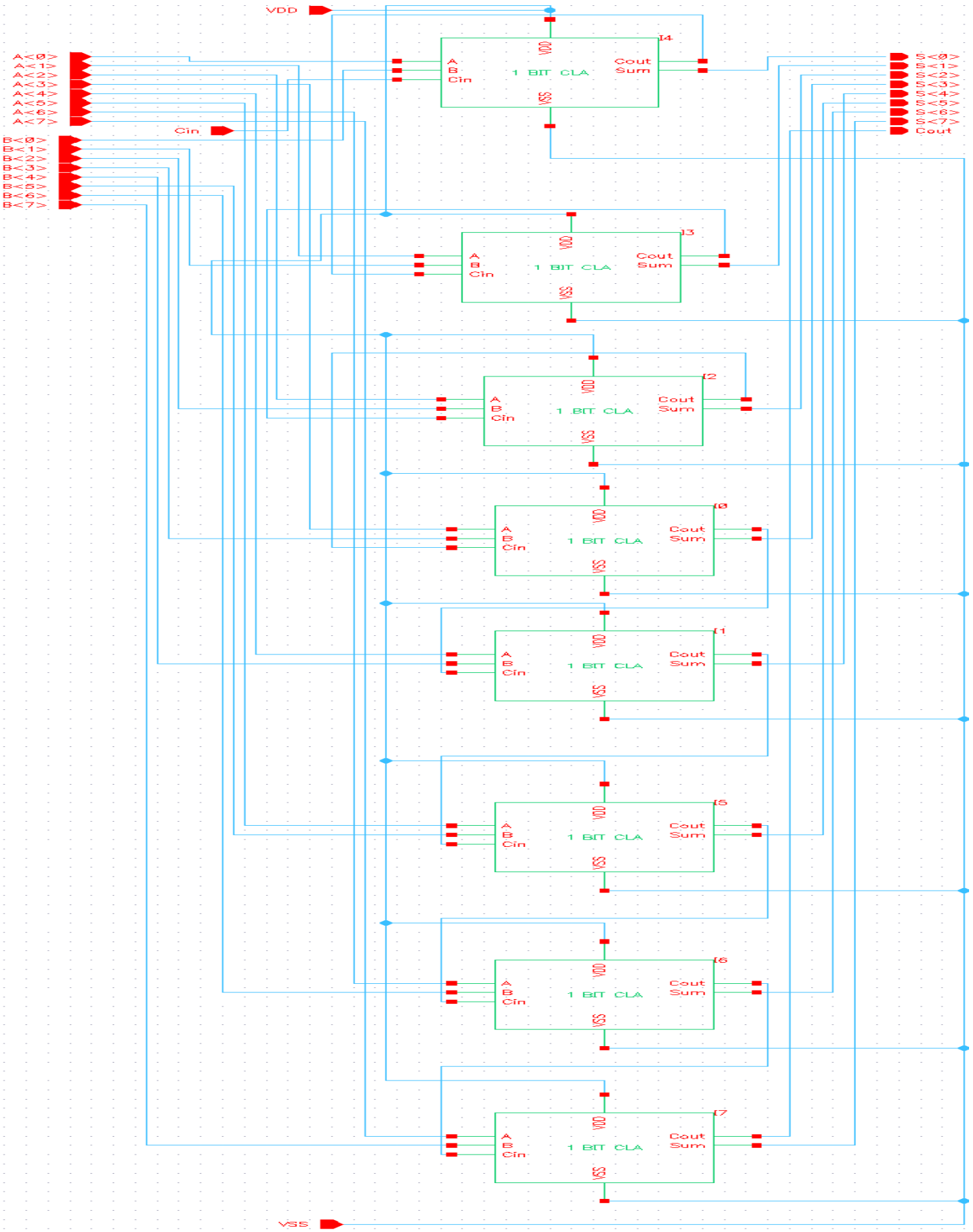
Logic

$$P_i = A_i \oplus B_i \; Gi = A_i B_i \; C_i = G_i + P_i C_{in} Si = C_{in} \oplus P_i$$

Schematic 2.1.a: Bit-Level Adder Implementation



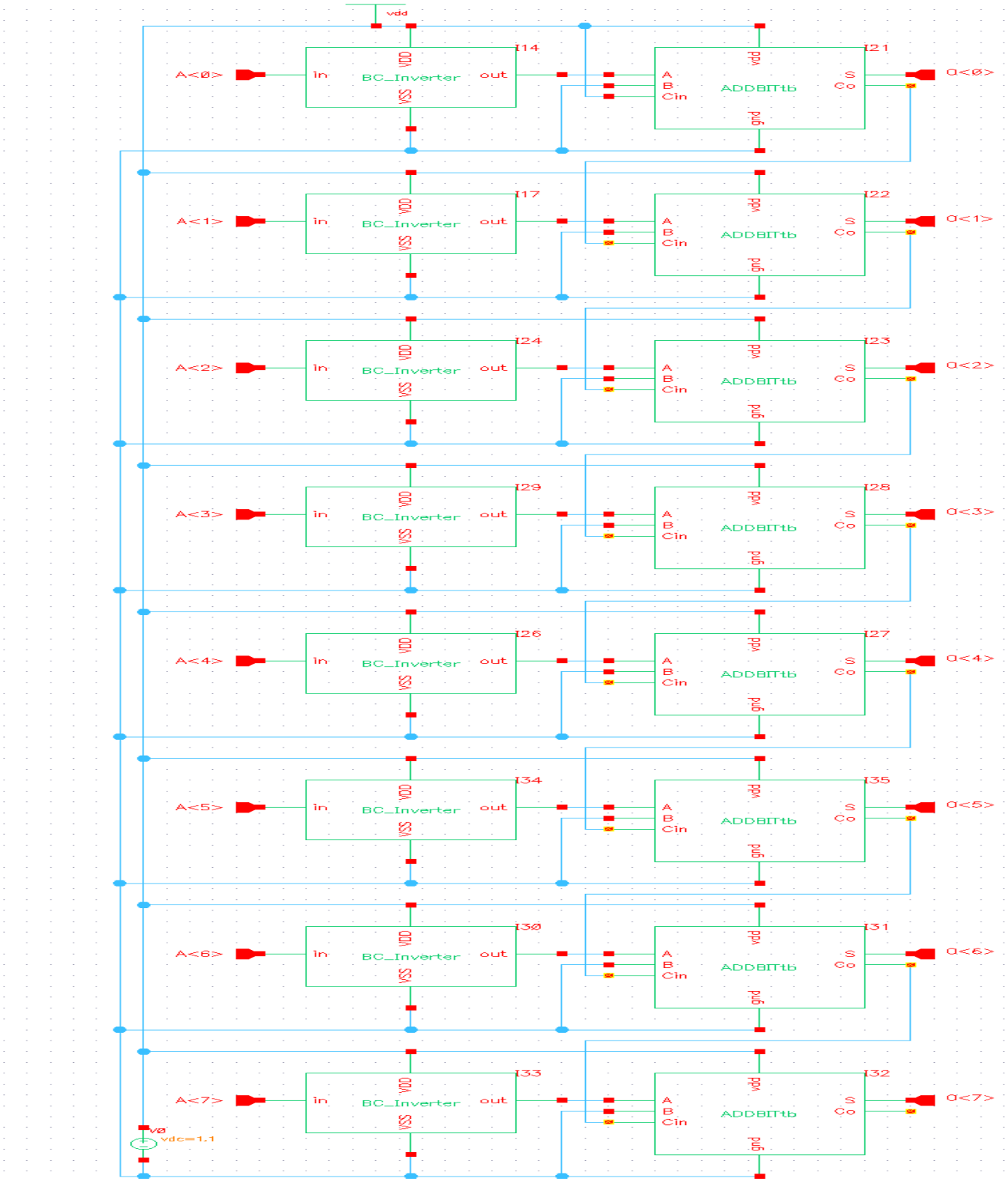
Schematic 2.1.b: Adder Byte-Level Hierary



Appendix A.2.2: implementation schematics – 2's Comp.

The implementation of a 2's complement follows directly from the addition operator, with minor modifications, as the logical expression for a 2's complement of a binary number is merely a bitwise inversion of the original + 1. This can conveniently be mapped to cin0 on the adder, suppressing all b.

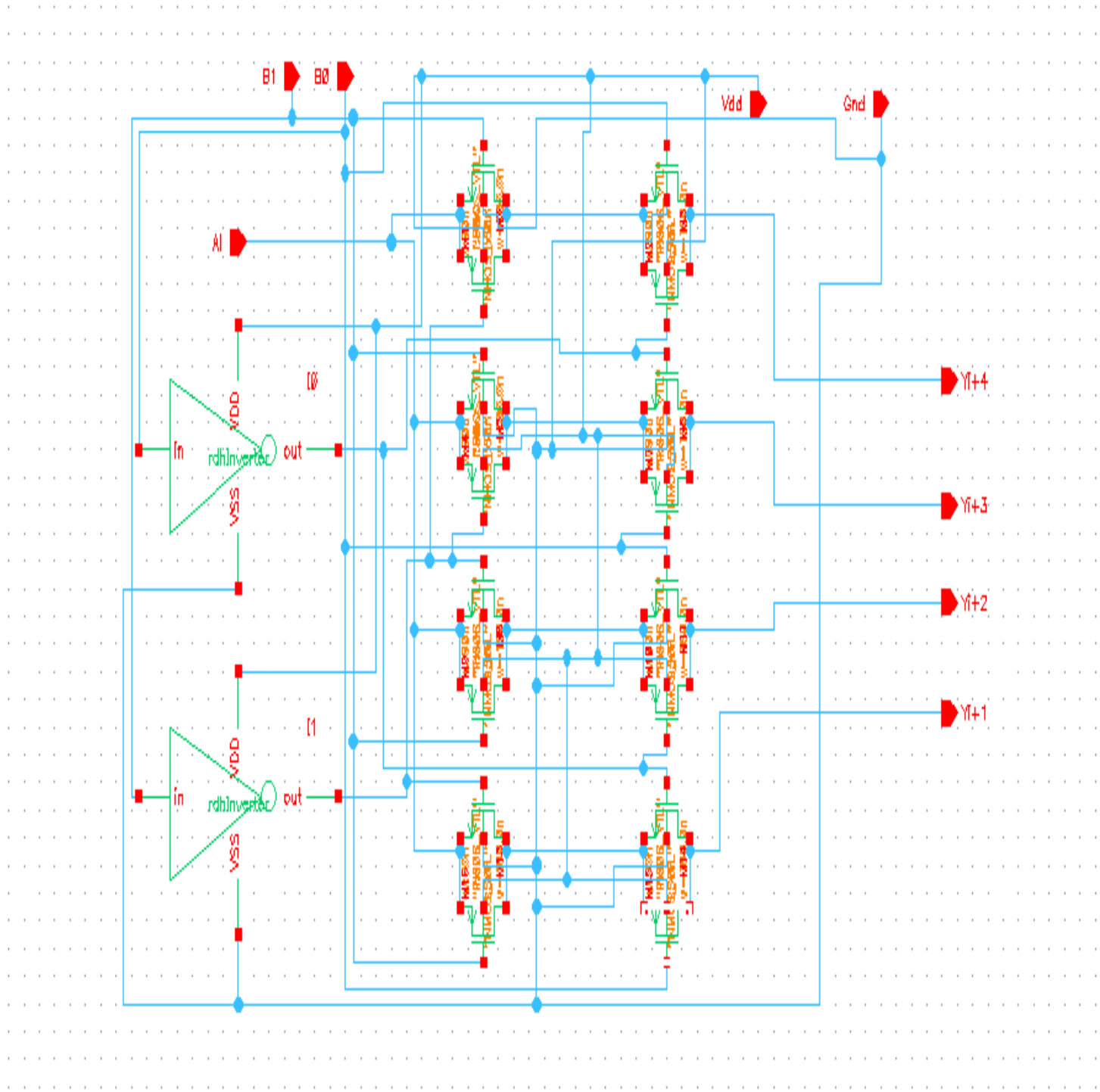
Schematic 2.2.a: 8-Bit hierarchy for 2COMP function



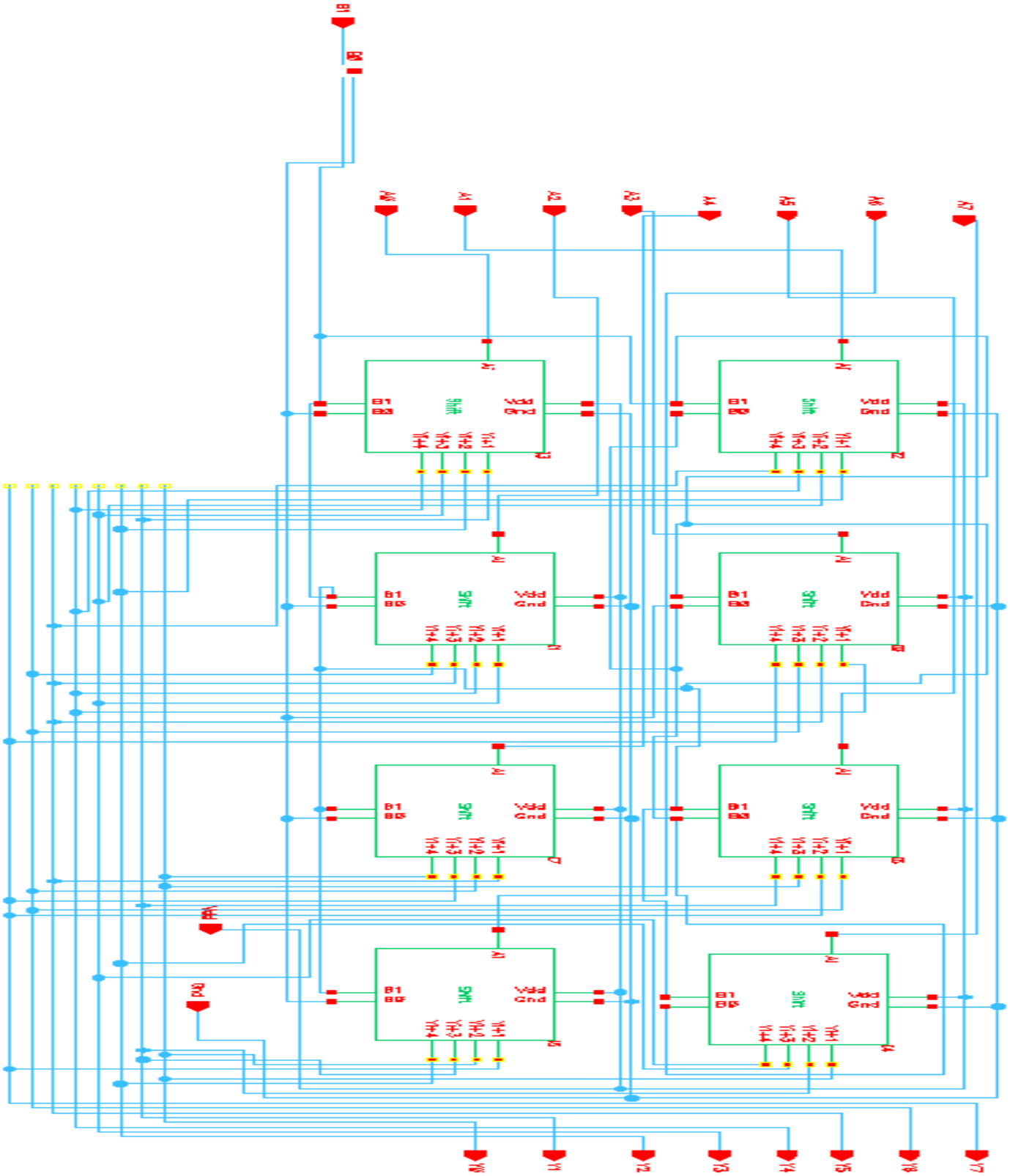
Appendix A.2.3: Implementation Schematics - Shift

For the 8-bit shift block, we decided to create a cleaner and more simplified implementation of a pass-transistor logic barrel shifter utilizing eight separate single-bit shift blocks. These blocks are capable of accepting one input and shifting that value by up to 4 bits. Then, these bit shift blocks were used in creating a full 8-bit barrel shifter by properly connecting each input to a particular bit shift block and the outputs of each block to the according output pins of the entire block itself.

Schematic 2.3.a: Bit-Level Shift Implementation



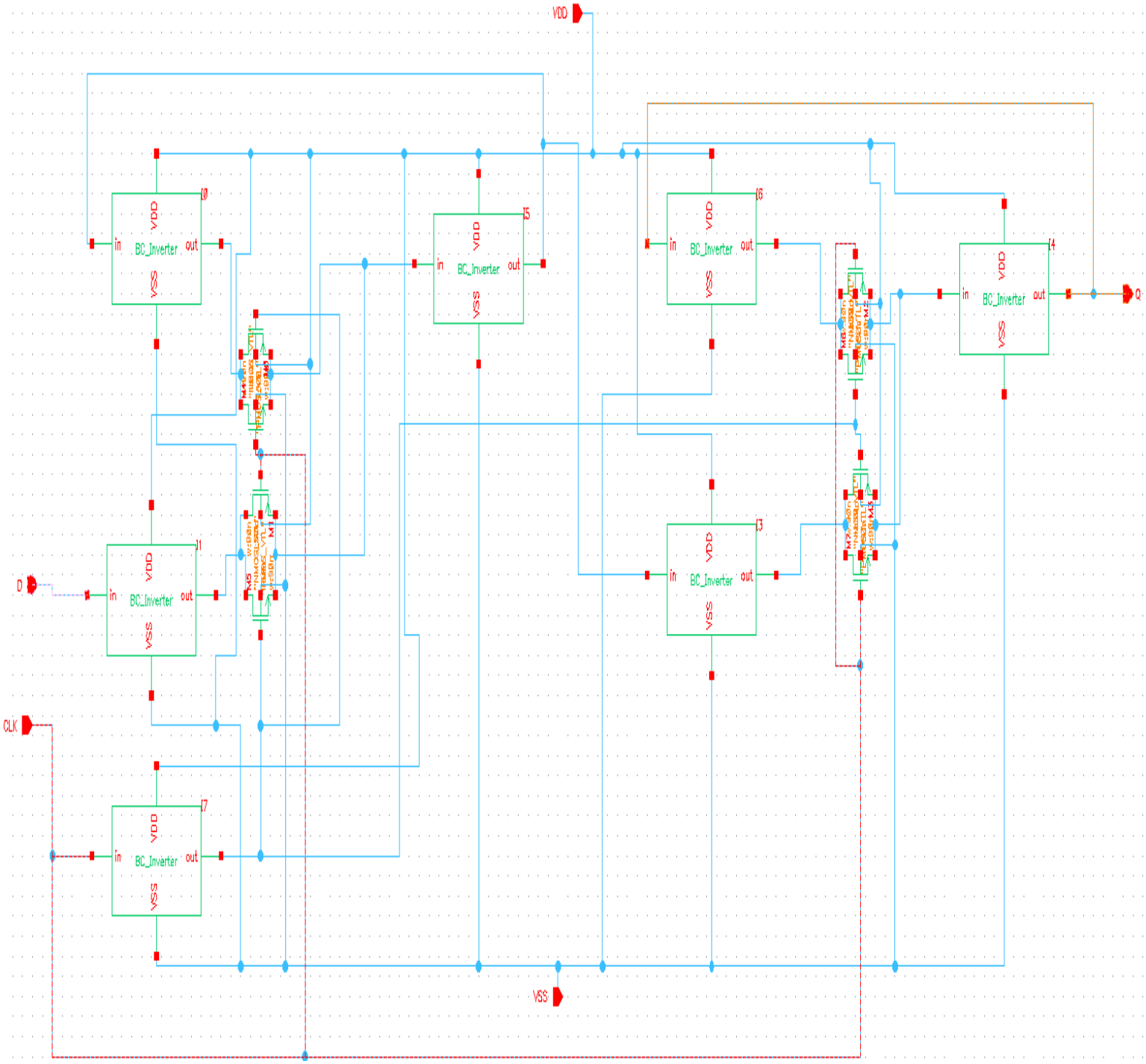
Schematic 2.3.b: **Shift Byte-Level Hierarchy**



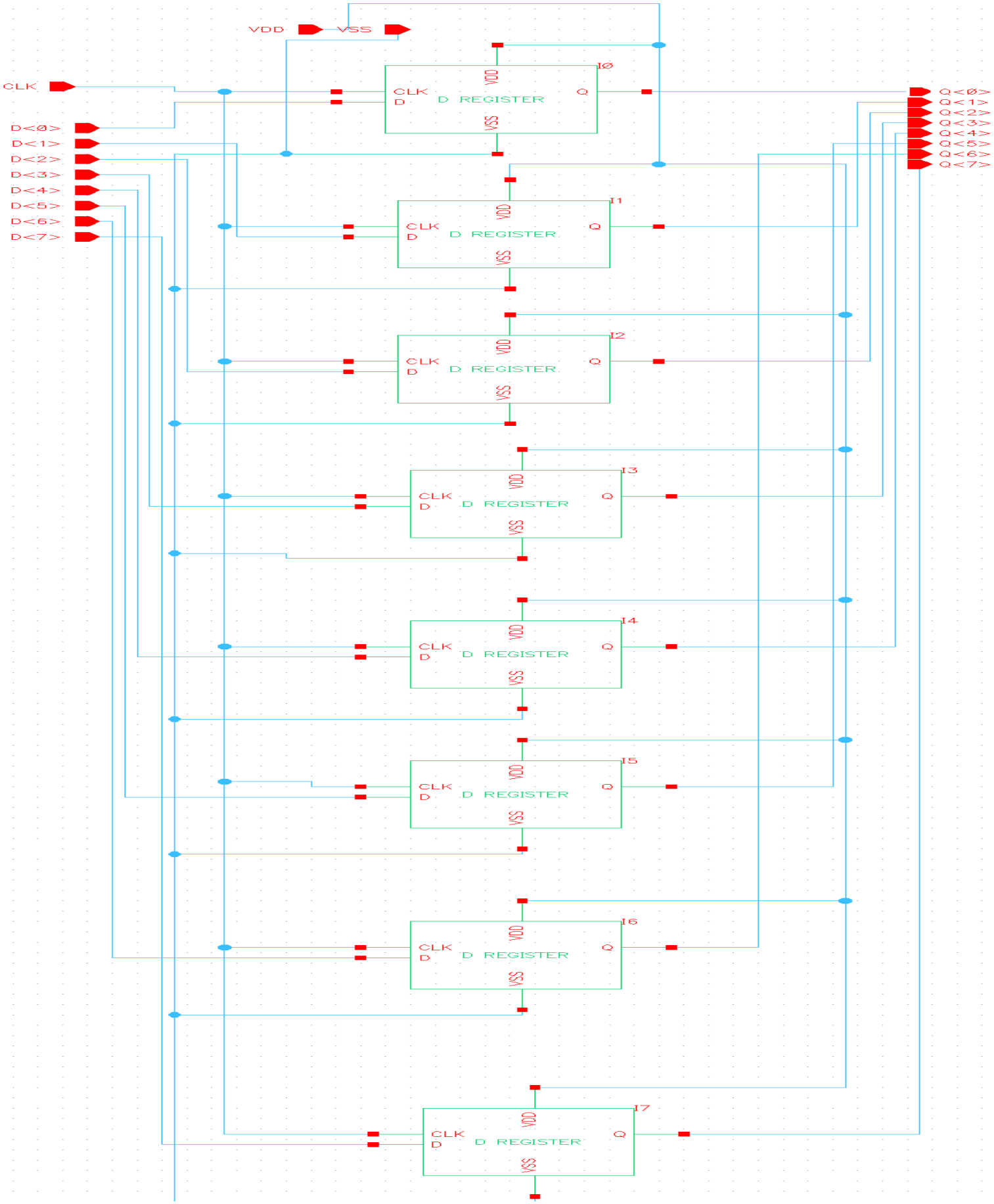
Appendix A.2.4: Implementation Schematics – Register

For the Register, we chose to use a simple Master-Slave Positive Edge Triggered Register, constructed with 2 D Flip Flops. We chose to use Pass Transistor Logic for the same reasons as the Adder, namely reduced input load and a lower transistor count.

Schematic 2.4.a: Bit-Level Register Implementation

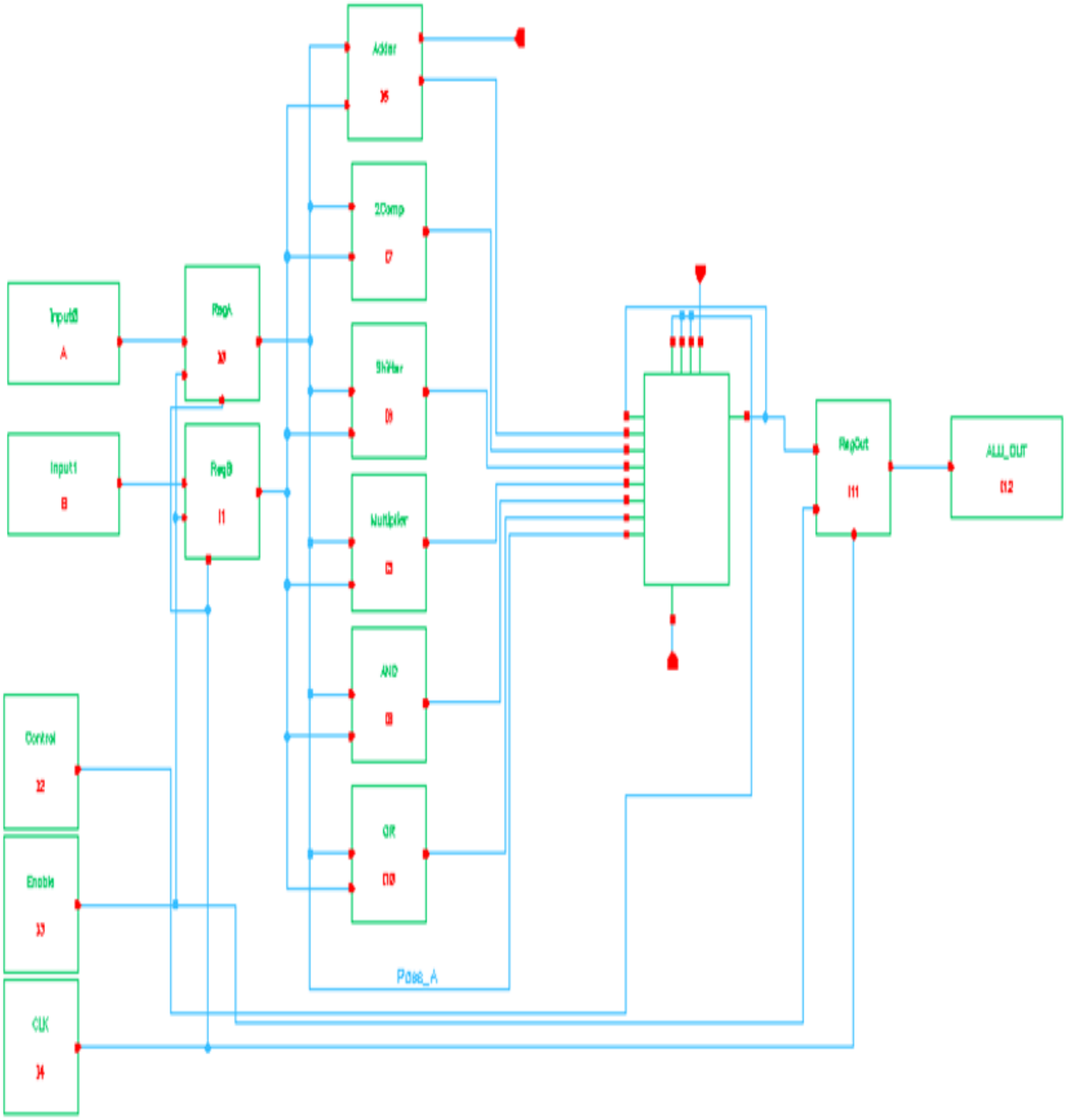


Schematic 2.4.b: **Register Byte-Level Hierarchy**



Appendix A.2.5: Implementation Schematics – Top-Level Connections

Schematic 2.5: **Top-Level Block Connection Diagram**



Appendix B: Transient Analysis and Testing Strategy

B.2.1: ADD

Graph 2.1.a: Proof of Logical Functionality - Single Bit ADD
Graph 2.1.b: Estimation of worst case delay - ADD

B.2.2: 2COMP

Graph 2.2.a: Estimation of worst case delay – 2Comp

B.2.3: SHIFT

Figure 2.3: Test Bench Setup for Shifter Analysis
Graph 2.3.a: Plot of Shifter Transient Analysis Input
Graph 2.3.a: Plot of Shifter Transient Analysis Output

B.2.4: Register

Graph 2.4.a: Proof of Logical Functionality – Register

B.2.5: Worst Case Delay

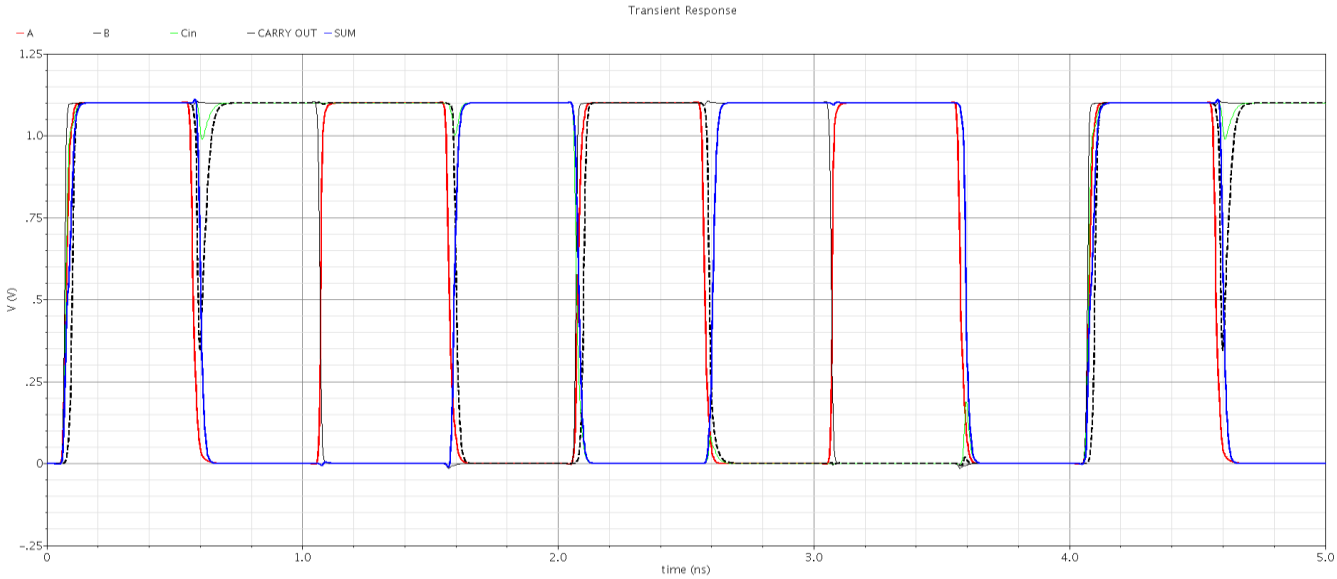
Table 2.5: Per Operation Worst-Case Function Delay

Appendix B.2.1: Add Transient Analysis

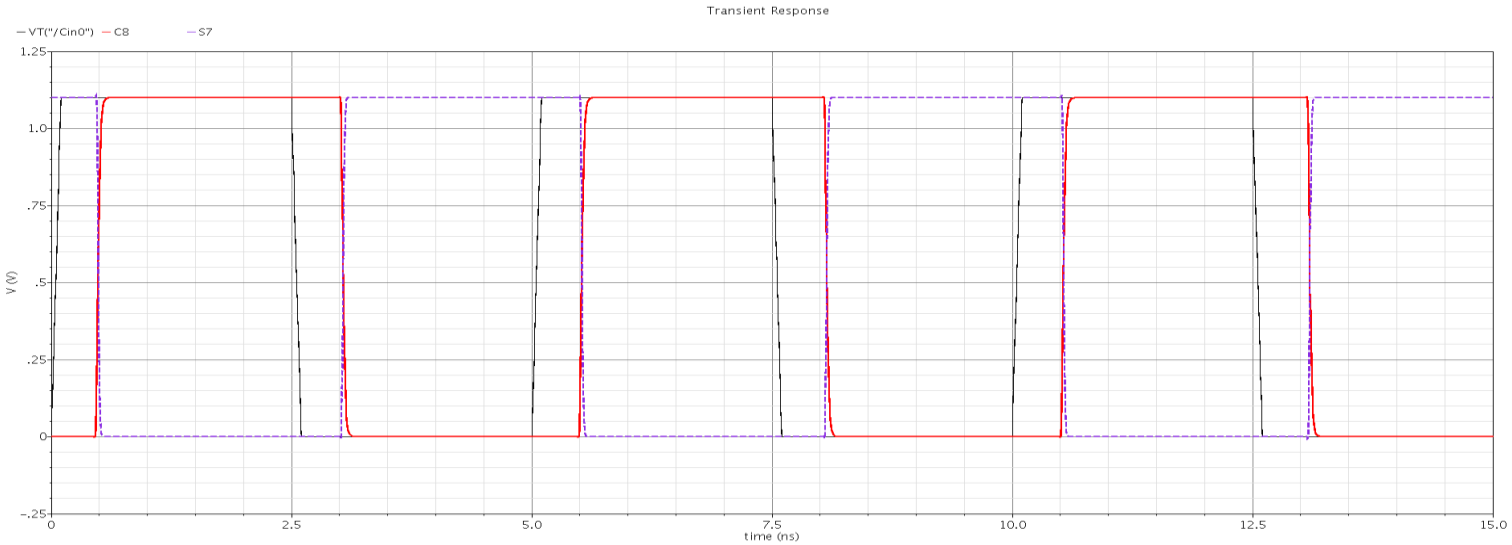
Two simulations were done to test the adder. First, a trial of all possible inputs to a single full adder were performed and the resulting outputs were plotted (2.1.a). This proved that the logic for the block was sound, and because the full 8 bit adder is composed of multiples of this same exact block, an analysis of the full 8 bit adder for every possible input (an extremely tedious task) was unnecessary. The second simulation attempted to find the worst case delay for the Adder. This had to be done on the full 8 bit design because the worst case delay for an 8 bit adder is on the most significant carry computation. To attempt to find the worst case delay, we came up with a scenario in which all of the outputs must change state. In this scenario, all of the "A" bits were set to zero, Carry In was set to 1, and all of the B bits were set to 1. Thus the sum would be 00000000 with a carry out of 1. Then, the carry in was changed to 0, causing the sum to change to 11111111 with a carry out of 0. This was postulated to cause the most delay possible. (2.1.b)

Graph 2.1.a: Proof of Logical Functionality - Single Bit ADD

111 110 101 100 011 010 001 000



Graph 2.1.b: Estimation of worst case delay – ADD

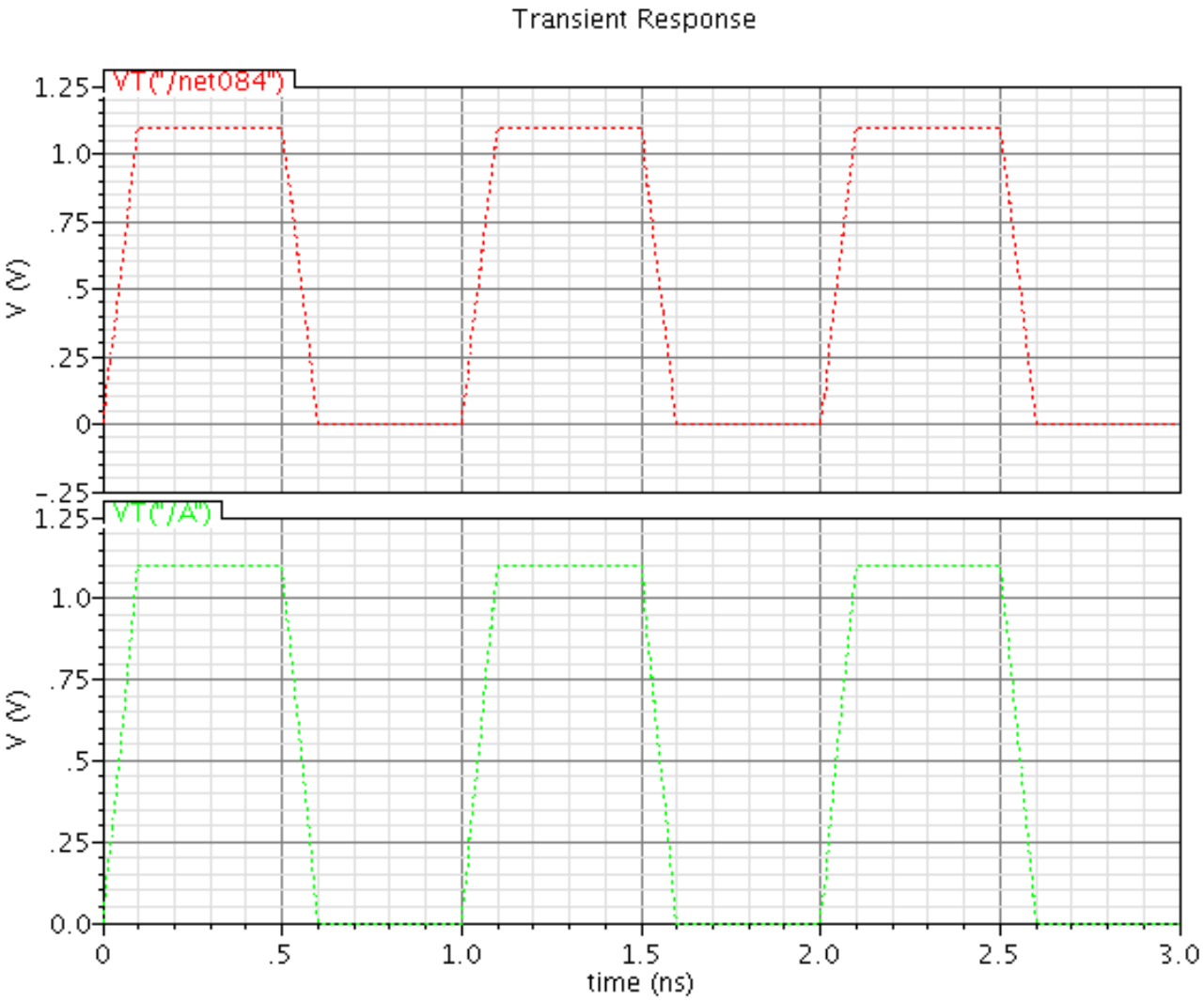


Appendix B.2.2: 2's Complement Transient Analysis

By design, logical functionality follows implicitly from the proof of the add function, as 2's complement reduces to a special case of add as a generic operator (or more properly a subset) and thus is invariant with respect to logical operation.

To resolve our 2's complement delay, a similar situation was constructed as in the adder, only with the addition of the inversion on the input and under the constraints (B=00000000, Cin0=1)

Graph 2.2.a: Estimation of worst case delay – 2Comp



Appendix B.2.3: Shifter Transient Analysis

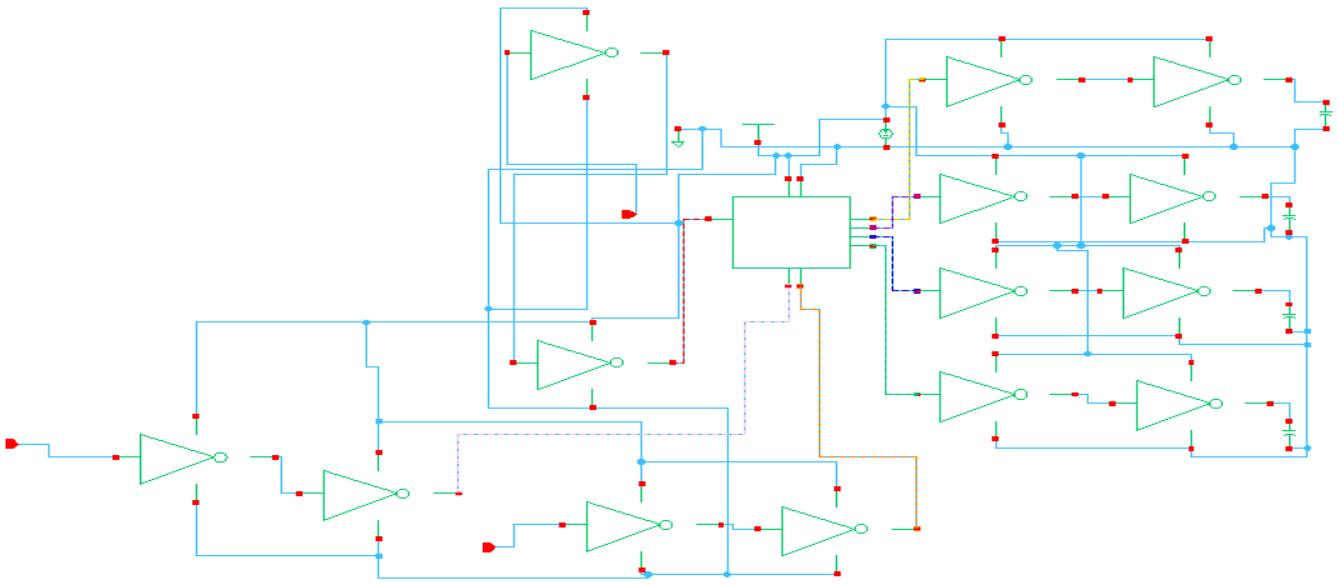
For simplicity in demonstrating the function of the shift block, we ran simulations only on the core building block of the design, which is the single-bit shifter. The simulation attempted to prove that a particular input value Ai (i.e., 0 or 1) was shifted to the appropriate output pin (denoted Yi+1, Yi+2, Yi+3, and Yi+3). All inputs were driven by two series inverters, and all outputs were loaded by two series inverters. The test bench for this setup is shown below (fig. 2.3).

We encountered some unexpected waveforms during the simulation of this shift block. We first went back to the schematic level design of the shift block to troubleshoot. The block design itself is based on a PTL topology, as shown in the preceding section. We ensured that the transistor widths were set correctly (i.e., in the parallel NMOS-PMOS configuration of each transmission gate, the PMOS width was 2x that of the NMOS), all gates were tied to the proper control inputs (B1 and B0, or their complements) depending on the shift (i.e., 1 bit, 2 bits, etc.), and that all devices were receiving the proper power supply. We then began to investigate our test bench setup to ensure that it was appropriate, with two series inverters to drive inputs and load outputs. We were not able to determine what was causing the unexpected waveforms on the outputs during the transient simulation, but are continuing to investigate the cause.

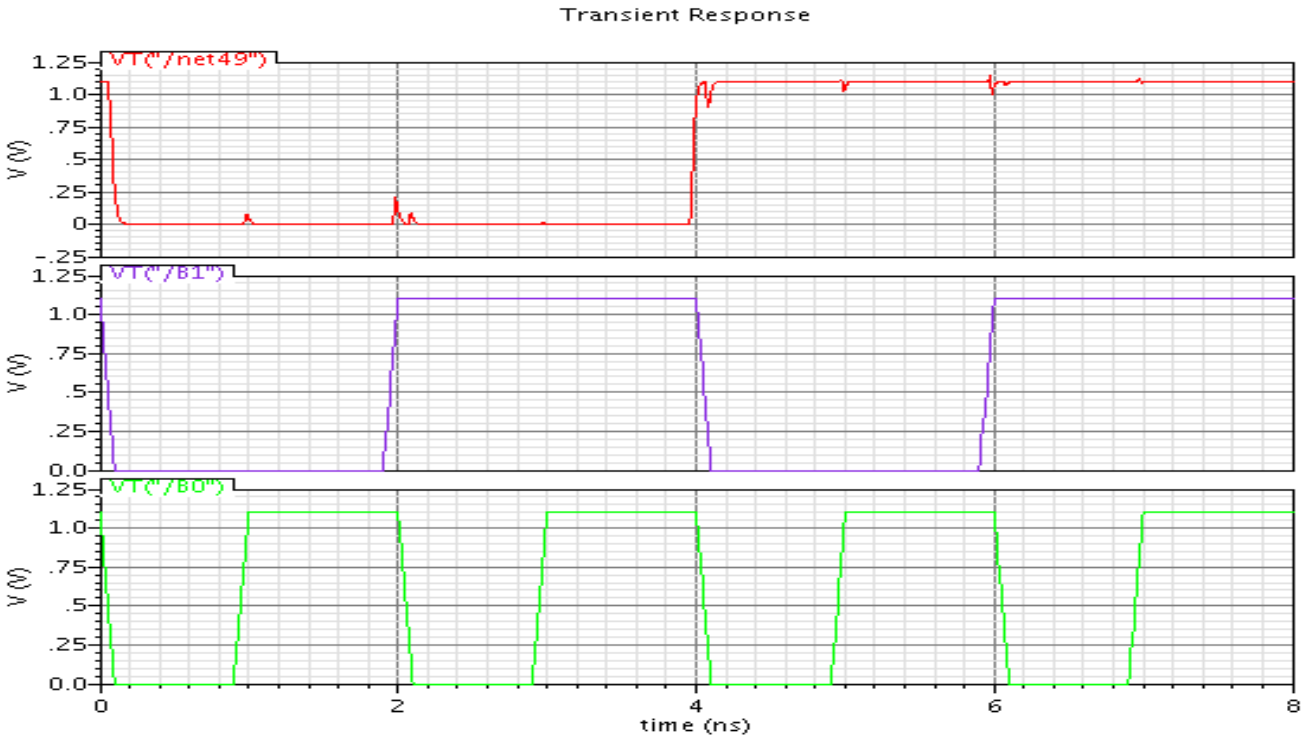
Below are separate plots of the inputs (2.3.a) and the outputs (2.3.b). The inputs were Ai (red), B1 (purple), and B0 (green). For this simulation, to prove that the bits shifted properly, we added pulse stimuli on all three inputs and were careful to choose the pulse widths so that all possible combinations were tested in one simulation (in particular: B1B0 = 00, 01, 10, 11 for Ai = 0 and Ai = 1).

The outputs shown are Yi+1 (red), Yi+2 (blue), Yi+3 (orange), and Yi+4 (green). Our main concerns are with the bottom two waveforms, representing Yi+3 and Yi+4. First, it seems that Yi+3 begins rises to approximately half of the full logic level (1.1 V) at B1B0 = 01, and then to the full logic level at B1B0 = 10 as it is supposed to. Also, Yi+4 is going to 1 at B1B0 = 10, prior to when it is supposed to at B1B0 = 11, and is going to 0 at B1B0 = 10, prior to when it is supposed to at B1B0 = 11. The rest of the waveforms, however, appear to demonstrate proper functionality of the shift block, with the output pin receiving the shifted input value (Ai) according to the control input (B1B0), as seen below.

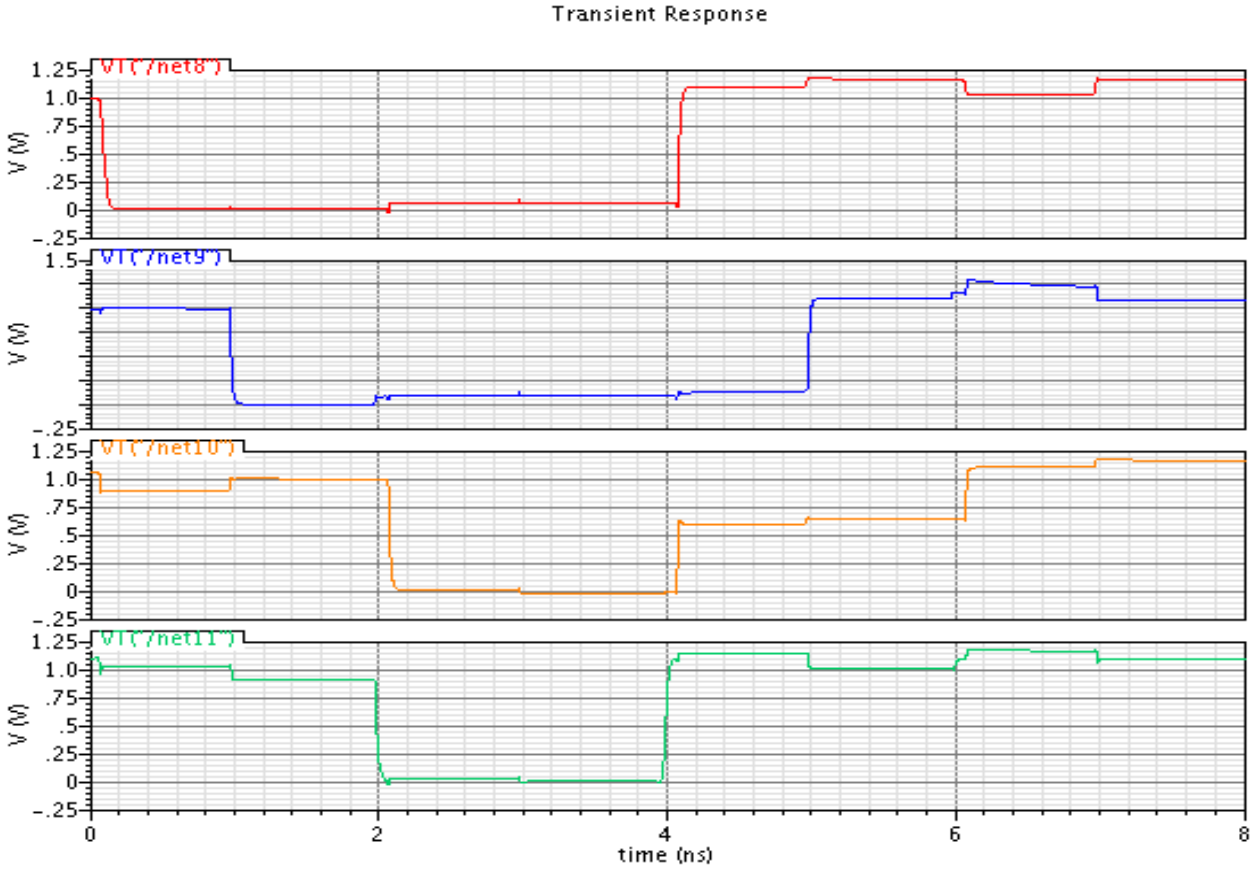
Figure 2.3: Test Bench Setup for Shifter Analysis



Graph 2.3.a: Plot of Shifter Transient Analysis Input



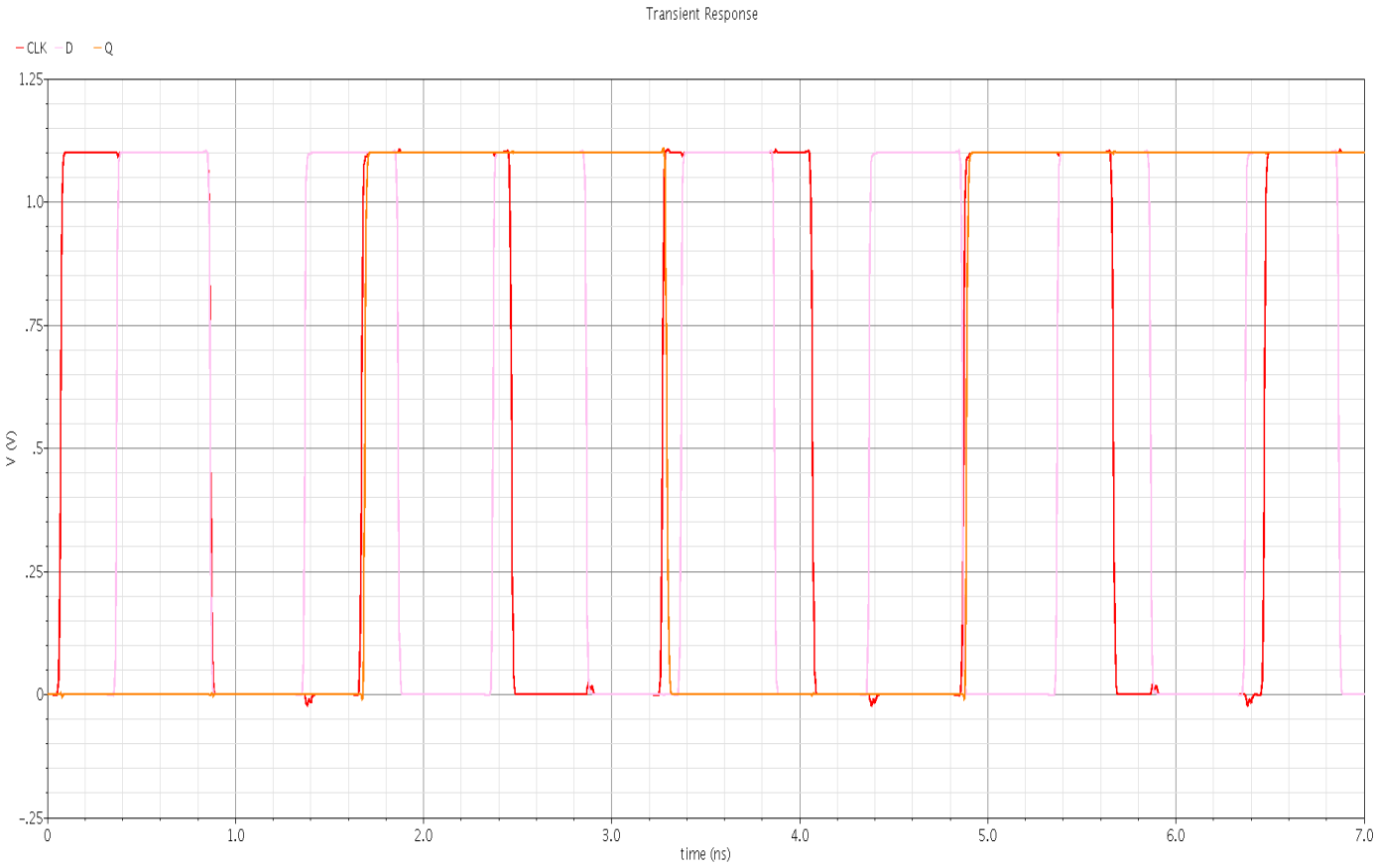
Graph 2.3.b: Plot of Shifter Transient Analysis Output



Appendix B.2.4: Register Transient Analysis

Our register is an edge triggered Master-Slave Register, so to verify that it works as designed, all we had to do was ensure that whenever a bit is stored, it is maintained for 1 full clock cycle, and that it changes correctly. We did this by changing the input (D) at different times throughout the clock cycle to verify that the output did not vary throughout the clock cycle, and that whatever value is present at the input is the value maintained for that clock cycle. This is demonstrated below as proof that the register works.

Graph 2.4.a: Proof of Logical Functionality – Register



Appendix B.2.5: Worst Case Delay

Table 2.5: Per Operation Worst-Case Function Delay

Operation	Worst-Case Delay	Conditions for Worst-Case
ADD	0.5 nS	A=00000000 B=11111111 Cin starts at 1, then goes to 0
2COMP	0.65 nS	A=00000001; as would be with the adder, except an input all low excluded as it doesn't change the output
SHIFT	72 pS	Time for output pin to reflect shifted input value Ai upon receiving valid control input B1B0
AND	18.7 pS	Both inputs go from low to high.
OR	112 pS	Low to high transition of output when both inputs go from low to high
Pass A	3.46 pS	Input sweep from low to high